A Generalized Quantifier Concept in Computational Complexity Theory

Heribert Vollmer

Theoretische Informatik
Universität Würzburg
Am Exerzierplatz 3
D-97072 Würzburg, Germany
vollmer@informatik.uni-wuerzburg.de

Abstract

A notion of generalized quantifier in computational complexity theory is explored and used to give a unified treatment of leaf language definability, oracle separations, type 2 operators, and circuits with monoidal gates. Relations to Lindström quantifiers are pointed out.

Keywords: computational complexity, computation model, logic in computer science, finite model theory, generalized quantifier

1 Introduction

In this paper we develop a unified view at some notions that appeared in computational complexity theory in the past few years. This will be in the form of operators transforming complexity classes into complexity classes. Each such operator is given in the form of a quantifier on strings. This will immediately subsume as special cases the well known universal, existential, and counting quantifiers examined in various complexity theoretic settings [SM73, Wra77, Wag86b, Wag86a, Tor91]. But also a lot of constructions from other subareas of complexity theory can best be understood in terms of such operators. These include circuits with arbitrary monoidal gates [BIS90, BI94], oracle operators [BW96, BVW96], leaf languages (introduced in [BCS92, Ver93] and examined for different computation models in [HLS$^+$93, JMT94, CMTV98]). We survey some results from these areas and establish some new connections.

In finite model theory, examinations of the expressive power of various logics enhanced by Lindström quantifiers form a very well established field of active research. Descriptive complexity theory has characterized a great bulk of complexity classes by such logics. We will show that classes defined by our general operator can in a uniform way be characterized by model theoretic means using Lindström quantifiers.

In the following, we assume some familiarity of the reader with basic formal language theory (refer to [RS97]), basic complexity classes and resource-bounded reducibilities (refer to the standard literature, e.g. [Pap94, BC94, BDG95]; all complexity classes that appear in this paper without definition are defined in [Joh90]), as well as with the basics of finite model theory (refer to [Vää94, EF95]).

2 Definition

Given a language A over some alphabet Σ, we denote the characteristic function of A by χ_A, i.e. for all $x \in \Sigma^*$, $\chi_A(x) = 1$ if $x \in A$, and $\chi_A(x) = 0$ otherwise.

We will always assume some order on the alphabets we use; therefore it makes sense to talk about the lexicographic order $\prec$ of Σ^*, and for $x, y \in \Sigma^*$, $x \preceq y$, we define the characteristic string of A from x to y as $\chi_A[x \dots y] =_{\text{def}} \chi_A(x)\chi_A(x+1)\cdots\chi_A(y)$. Here, $x+1$ denotes the successor of x. In fact, we will presuppose an underlying bijection between Σ^* and the set $\mathbb{N}$ of natural numbers, and we use the notation $\chi_A[i \dots j]$ for $i, j \in \mathbb{N}$.

Let $\langle \cdot, \cdot \rangle$ denote a standard pairing function. For a set $A \subseteq \Sigma^*$ and a string $x \in \Sigma^*$, define $A_x =_{\text{def}} \{\, y \mid \langle x, y \rangle \in A \,\}$.

Looking now at the well-known characterization of the polynomial hierarchy by polynomially length-bounded universal and existential quantifiers $\exists^p$, $\forall^p$ [Wra77], the following is clear:

- A language L is in NP if and only if there is a language $A \in \mathrm{P}$ and a function f computable in polynomial time such that for all x,

$$x \in L \iff \chi_{A_x}[0 \dots f(x)] \in (0+1)^*1(0+1)^* \text{ (i.e., contains a “1”).}$$

- A language L is in coNP if and only if there is a language $A \in \mathrm{P}$ and a polynomial p such that for all x,

$$x \in L \iff \chi_{A_x}[0 \dots f(x)] \in 1^* \quad \text{(i.e., consists out of “1”s only).}$$

An analogous result holds for the class PP, which was characterized in [Wag86a] in terms of the so called polynomially length-bounded counting quantifier C^p:

- A language L is in PP if and only if there is a language $A \in \mathrm{P}$ and a function f computable in polynomial time such that for all x,

$$x \in L \iff \chi_{A_x}[0 \dots f(x)] \text{ contains more “1”s than “0”s.}$$

The class US (for unique solution) is defined by polynomial time nondeterministic Turing machines M which accept an input x if and only if there is exactly one accepting path in the computation tree of M on x.

- A language L is in US if and only if there is a language $A \in \mathrm{P}$ and a function f computable in polynomial time such that for all x,

$$x \in L \iff \chi_{A_x}[0 \dots f(x)] \in 0^*10^*.$$

Thus we see that here the semantics of quantifiers is defined by giving languages over the binary alphabet ($E =_{\text{def}} (0+1)^*1(0+1)^*$ for $\exists^p$, $U =_{\text{def}} 1^*$ for $\forall^p$, and $\text{maj} =_{\text{def}} \{ w \in \{0,1\}^* \mid w \text{ contains more "1"s than "0"s} \}$ for C^p). The following generalization now is immediate:

Let $B \in \{0,1\}^*$, let $\mathcal{K}$ be a class of sets, and $\mathcal{F}$ be a class of functions from $\Sigma^* \to \mathbb{N}$. Define the class $(B)^{\mathcal{F}}\mathcal{K}$ to consists of all sets L for which there exist some $A \in \mathcal{K}$ and some function $f \in \mathcal{F}$ such that for all $x \in \Sigma^*$, $x \in L \iff \chi_{A_x}[0 \ldots f(x)] \in B$.

We use the following shorthands: Write $(A)^{\mathrm{p}}\mathcal{K}$ ($(A)^{\log}\mathcal{K}$, $(A)^{\mathrm{plog}}\mathcal{K}$, resp.), if $\mathcal{F}$ is the class of all functions from $\Sigma^* \to \mathbb{N}$ computable in polynomial time (logarithmic time, polylogarithmic time, resp.) on deterministic Turing machines (i.e., $\mathcal{F} = \mathrm{FP}$, $\mathcal{F} = \mathrm{FDLOGTIME}$, $\mathcal{F} = \mathrm{FPOLYLOGTIME}$, resp.). For sub-linear time bounds we use Turing machines with index tape and random access to their input, working in the unrestricted mode (for background, refer to [CC95, RV97]). Observe that a function $f \in \mathrm{FP}$ is polynomially length-bounded, $f \in \mathrm{FDLOGTIME}$ is length-bounded by some function $c \cdot \log n$, and $f \in \mathrm{FPOLYLOGTIME}$ is polylogarithmically length-bounded. If $\mathcal{L}$ is a class of languages, then $(\mathcal{L})^{\mathcal{F}}\mathcal{K} =_{\text{def}} \bigcup_{B \in \mathcal{L}} (B)^{\mathcal{F}}\mathcal{K}$.

If we take the above three languages E, U, and maj, and look at different function classes $\mathcal{F}$, we get the existential, universal, and counting quantifier for various length-bounds.

The above definition appeared in [Vol96b] and (for the special case $\mathcal{F} = \mathrm{FP}$) in [BS97].

3 Polynomial Time Leaf Languages

The most examined special case of our general operator is probably the polynomial time case, i.e. the base class $\mathcal{K}$ is the class P (and $\mathcal{F} = \mathrm{FP}$). In this case there is a very intuitive way of visualizing the operator via so called *leaf languages*.

3.1 Definition

In the leaf language approach to the characterization of complexity classes, the acceptance of a word input to a nondeterministic machine depends only on the values printed at the leaves of the computation tree. To be more precise, let M be a nondeterministic Turing machine, halting on every path printing a symbol from an alphabet Σ, with some order on the nondeterministic choices. Then, $\text{leafstring}^M(x)$ is the concatenation of the symbols printed at the leaves of the computation tree of M on input x (according to the order of M's paths given by the order of M's choices). Given now a language $B \subseteq \{0,1\}^*$, we define $\text{Leaf}^M(B) = \{ x \mid \text{leafstring}^M(x) \in B \}$.

Call a computation tree of a machine M *balanced*, if all of its computation paths have the same length, and moreover, if we identify every path with the string over $\{0,1\}$ describing the sequence of nondeterministic choices on this path, then there is some string z such that all paths y with $|y| = |z|$ and $y \preceq z$ (in lexicographic ordering) exist, but no path y with $y \succ z$ exists.

A leaf language $B \subseteq \Sigma^*$ now defines the class $\text{BLeaf}^{\text{P}}(B)$ of all languages L for which there exists a nondeterministic polynomial time machine M whose computation tree is always balanced, such that $L = \text{Leaf}^M(B)$. Let $\mathcal{C}$ be a class of languages. The class $\text{BLeaf}^{\text{P}}(\mathcal{C})$ consists of the union over all $B \in \mathcal{C}$ of the classes $\text{BLeaf}^{\text{P}}(B)$.

This computation model was introduced by Bovet, Crescenzi, and Silvestri, and independently Vereshchagin [BCS92, Ver93] and later examined by Hertrampf, Lautemann, Schwentick, Vollmer, and Wagner [HLS+93], and Jenner, McKenzie, and Thérien [JMT94], among others. See also the textbook [Pap94, pp. 504f].

Jenner, McKenzie, and Thérien also considered the case where the computation trees are not required to be balanced. For that case, let B be any language. Then, the class $\text{Leaf}^{\text{P}}(B)$ consists of those languages L for which there exists a nondeterministic polynomial time machine M without further restriction, such that $L = \text{Leaf}^M(B)$. Let $\mathcal{C}$ be a class of languages. The class $\text{Leaf}^{\text{P}}(\mathcal{C})$ consists of the union over all $B \in \mathcal{C}$ of the classes $\text{Leaf}^{\text{P}}(B)$. (Strictly speaking, the definition of *balanced* given in [JMT94] is different from ours and, at first sight, slightly more general. However, it is easy to see that both definitions are equivalent.)

The reader now might wonder about the seemingly unnatural condition that the nondeterministic choices of M are ordered. In fact, most complexity classes of current focus can be defined without this assumption—in this case the leaf language B has the special property that we can permute the letters in a given word without affecting membership in B. (Cf. our results on *cardinal languages* in Sect. 3.3.2 below.) However, strange classes where the order of the paths is important for their definition are conceivable, and the results presented below, especially the oracle separation criterion (Theorem 3.8), also hold for these pathologic cases.

The following connection to our generalized quantifier now is not too hard to see.

Theorem 3.1. *Let $B \subseteq \{0,1\}^*$. Then $(B)^{\text{p}}\text{P} = \text{BLeaf}^{\text{P}}(B)$.*

Proof sketch. ($\subseteq$) Let $L \in (B)^{\text{p}}\text{P}$, $x \in L \iff \chi_{A_x}[0 \ldots f(x)] \in B$. The nondeterministic machine, given x, branches on all possible second inputs y in the range $0, \ldots, f(x)$, and outputs $\chi_A(x, y)$.

($\supseteq$) Let $L \in \text{BLeaf}^{\text{P}}(B)$ via the nondeterministic machine M. Computation paths of a nondeterministic machines can be followed in polynomial time if the nondeterministic choices are known. Defining A to consist of all pairs (x, p) such that p is a sequence of nondeterministic choices leading to a path of M that outputs "1" and $f(x)$ to be the number of paths of M on input x, we have $x \in L \iff \chi_{A_x}[0 \ldots f(x)] \in B$. ❑

The definition of leaf languages allows for languages B not necessarily over the binary alphabet. If we want to come up with a connection to our generalized quantifier also for such B, we face a problem. In the definition in Sect. 2 the binary alphabet seems essential. Fortunately, for every B there is usually a $B' \subseteq \{0,1\}^*$ such that $\text{BLeaf}^{\text{P}}(B) = \text{BLeaf}^{\text{P}}(B')$, where B and B' are of the same complexity. In most cases, B' can simply be obtained from B by block encoding (then B and B' are FO-equivalent). We come back to this point in the next subsection.

3.2 The Complexity of a Leaf Language

In [HLS$^+$93] the question how complex a leaf language must be in order to characterize some given complexity class $\mathcal{K}$ was addressed. Let us start by considering some examples.

At great number of classes can be defined by regular leaf languages. This is obvious for NP, coNP and US as we saw in the previous section, for $\mathrm{Mod}_k\mathrm{P}$ (all words with a number of "1"s divisible by k), but also true for higher levels of the polynomial hierarchy (see below) and the boolean hierarchy over NP (e.g. the class NP$\wedge$coNP can be defined via the set of all words such that the string "010" appears at least once, but the string "0110" does not appear).

For other complexity classes, context-free languages come immediately to mind. PP can obviously be defined by the language maj from the previous section. Recalling the characterization of PSPACE via polynomial time alternating Turing machines, it is clear that the set of all (suitably encoded) boolean expressions involving the constants "true" and "false" and the connectives AND and OR that evaluate to "true", is an appropriate leaf language.

The question however arises if we can do better here. It was shown in [HLS$^+$93], that in the case of PSPACE there is a regular leaf language.

Let S_5 denote the word problem for the group of permutations on five elements (suitably encoded over the binary alphabet), i.e. S_5 consists of sequences of permutations which multiply out to the empty permutation.

Theorem 3.2. $(S_5)^{\mathrm{p}}\mathrm{P} = \mathrm{PSPACE}$.

Proof sketch. For the inclusion from left to right, just observe that a PSPACE machine can traverse the whole computation tree of a given nondeterministic machine to evaluate the product over S_5. This simulation then stops accepting if and only if the result is the identity permutation.

For the other direction, we are given a language $L \in$ PSPACE. Then there is a polynomial time alternating Turing machine accepting L. Thus, for every input w, machine M defines a polynomial depth computation tree $T(w)$ where the leafs carry values 0 or 1 and in the inner nodes the functions AND and OR are evaluated. $w \in L$ iff the root of this tree evaluates to 1. As a first step we transform this tree into a tree $T'(w)$ where in all the inner nodes the function NOR is evaluated. This can easily be achieved since the NOR function constitutes a complete basis for the boolean functions.

As a second step we now "simulate" NOR in S_5. This simulation is essentially due to David Barrington [Bar89]. Let b, c, d, e, f be the following permutations from the group S_5:

$$b = (23)(45), c = (12435), d = (243), e = (345), f = (152)$$

Further let a_0 be the empty permutation, denoted by $a_0 = ()$, and let $a_1 = (12345)$. Now consider the following product in S_5 including the variables x and y:

$$w(x, y) = a_0 b x^4 c y^4 d x e y f$$

Simple calculations show that $w(a_0, a_0) = a_1$ and $w(a_0, a_1) = w(a_1, a_0) = w(a_1, a_1) = a_0$. Thus coding the value *true* by a_1, and *false* by a_0, we can view w as the NOR-operation applied to x and y.

Now replace every appearance of a "NOR"-node in $T'(x)$ with sons x and y by a binary subtree of height 4 whose 16 leaves are

$$a_0 \;\; b \;\; x \;\; x \;\; x \;\; x \;\; c \;\; y \;\; y \;\; y \;\; y \;\; d \;\; x \;\; e \;\; y \;\; f$$

Thus we accept the input w if and only if the leaf string evaluates to a_1. Taking B to be the regular language

$$B =_{\text{def}} \left\{ x \;\middle|\; x \text{ is a string of elements from } S_5 \text{ which evaluates to } a_1 \right\},$$

we then get $\text{Leaf}^{\text{P}}(B) = \text{PSPACE}$. It is easy to go from B to the word problem S_5 (by just adding one more factor a_1^{-1}), and since we have an identity element which we can insert arbitrarily in the leaf string to fill gaps in the computation tree in order to make it balanced, we get $\text{PSPACE} = \text{Leaf}^{\text{P}}(B) = \text{BLeaf}^{\text{P}}(S_5) = (S_5)^{\text{p}}\text{P}$. ❑

The question now of course is what is so special about the language S_5. What can be said more generally? Using deep algebraic properties of regular languages exhibited in [Thé81, BT88] (see also the textbook [Str94]) one can show the following.

Let PH denote the union of all classes of the polynomial hierarchy [SM73], i.e. $\text{PH} = \text{NP} \cup \text{NP}^{\text{NP}} \cup \text{NP}^{\text{NP}^{\text{NP}}} \cup \cdots$. Let MOD-PH denote the oracle hierarchy constructed similarly, but now allowing as building blocks not only NP but also all classes Mod_kP for arbitrary $k \in \mathbb{N}$.

Theorem 3.3 [HLS⁺93].

1. *Let A be a regular language whose syntactic monoid is non-solvable. Then* $(A)^{\text{p}}\text{P} = \text{PSPACE}$.
2. *Let* SOLVABLE *denote the class of all regular languages whose syntactic monoid is solvable. Then* $(\text{SOLVABLE})^{\text{p}}\text{P} = \text{MOD-PH}$.
3. *Let* APERIODIC *denote the class of all regular languages whose syntactic monoid is aperiodic. Then* $(\text{APERIODIC})^{\text{p}}\text{P} = \text{PH}$.

Regular leaf languages for individual levels of the polynomial hierarchy can also be given. For example Σ_2^{p} can be defined over $\Sigma = \{a, b, c\}$ by $\Sigma^* c a^+ c \Sigma^*$, intuitively: "there is a block consisting out of 'a's only". This is an $\exists\forall$ predicate directly reflecting the nature of Σ_2^{p}-computations. If we now chose a simple block encoding this might lead us out of the aperiodic languages. However, we may proceed as follows: Define $A_2 = (0+1)^* 11 (010)^+ 11 (0+1)^*$. It is clear that this leaf language defines a subclass of Σ_2^{p}—just check that there are two substring 11 such that in between we have a sequence of occurrences of the 3-letter string 010; this is an $\exists\forall$ condition. On the other hand, suppose we are given a Σ_2^{p} machine M, i.e. an alternating machine with computation trees consisting of one level of $\exists$ nodes followed by a second level of $\forall$ nodes; i.e. the initial configuration is the root of an existential tree where in the leaves we append universal subtrees. We transform this into a tree where we use the substring "11" in the leafstring as separator between different $\forall$ subtrees, and within each such subtree we simulate an accepting path by the 3 leaf symbols "010" and a rejecting path by the symbol "0". Then M produces a tree with at least one universal subtree consisting out of only accepting paths iff the leaf word of this simulation is in A_2. Σ_3^{p} can similarly be defined via $A_3 = (0+1)^*111\overline{A_2}111(0+1)^*$. This generalizes to higher levels of the polynomial hierarchy. With some care one can show that A_2 and A_3 are in levels $\mathcal{B}_2$ and $\mathcal{B}_3$, resp., of the Brzozowski-hierarchy of regular languages. This hierarchy of star-free regular languages measures the nesting depth of the dot (i.e. concatenation) operation. For a formal definition see [Eil76]. More generally the following holds:

Theorem 3.4 [HLS$^+$93]. $(\mathcal{B}_k)^{\mathrm{p}}\mathrm{P}$ *is the boolean closure of the class* Σ_k^{p}.

Let us now come back to the question if PP (for which we gave a context-free leaf language above) can also be done by a regular language.

Corollary 3.5. PP *is not definable via a regular leaf language unless either* PP = PSPACE *or* PP $\subseteq$ MOD-PH.

Proof. If there is a regular leaf language L for PSPACE, then there are two cases to consider: either L is non-solvable (in this case PP = PSPACE) or L is solvable (then PP $\subseteq$ MOD-PH). ❑

In [HLS$^+$93] leaf languages defined by restricting resource bounds as time and space were examined. It was shown that the complexity class obtained in this way is defined via the same resource, but the bound is one exponential level higher, for example $(\mathrm{P})^{\mathrm{p}}\mathrm{P}$ = EXPTIME, $(\mathrm{NP})^{\mathrm{p}}\mathrm{P}$ = NEXPTIME, $(\mathrm{LOGSPACE})^{\mathrm{p}}\mathrm{P}$ = PSPACE, $(\mathrm{PSPACE})^{\mathrm{p}}\mathrm{P}$ = EXPSPACE, and so on. Denoting the levels of the alternating log-time hierarchy [Sip83] by $\Sigma_k^{\log}$ ($k \in \mathbb{N}$), we get the following special case:

Theorem 3.6. $(\Sigma_k^{\log})^{\mathrm{p}}\mathrm{P} = \Sigma_k^{\mathrm{p}}$.

3.3 Some Complexity Theoretic Applications

3.3.1 Normal Forms

The characterization of PSPACE (Theorem 3.2) was somewhat surprising, since it points out a very restricted normal form for PSPACE computations. Cai and Furst defined a class $\mathcal{K}$ to be $\mathcal{K}'$*-serializable*, if every $\mathcal{K}$ computation can be organized into a number of local computations $c_1, \ldots, c_r$ (which in turn are restricted to be $\mathcal{K}'$ computations), each passing only a constant number k of bits as the result of its computation to the next local computation. The sequence $c_1, \ldots, c_r$ is uniform in the sense that there is one $\mathcal{K}'$ program that gets as input only the original input, a number i, and a string of k bits, and computes the k-bit-result of c_i's computation. Please refer to [CF91] for a formal definition. Machines as just described are also called bottleneck machines. The *bottleneck* refers to the restricted way of passing information onwards.

Corollary 3.7 [HLS$^+$93]. PSPACE *is* AC^0*-serializable.*

Proof sketch. Let $L \in \mathrm{BLeaf}^{\mathrm{P}}(S_5)$ via machine M. The information passed from one computation to the next will be an encoding of an element of the group S_5. Each local computation uses its number to recover from it a path of the nondeterministic Turing machine. (If the number does not encode a correct computation path, then we simply pass the information we get from our left neighbor onwards to the right.) The leaf symbol on this path is then multiplied to the permutation we got from the left, and the result is passed on to the right. This can be done in AC^0 since computation paths of polynomial time Turing machines can be checked in AC^0. (A computation path consists not only out of M's nondeterministic choices, but is a complete sequence of configurations of M.) ❑

The power of bottleneck machines was examined in detail in [Her97]. He gave a connection between these machines and leaf languages defined via transformation monoids. The power of bottleneck machines as a function of the number of bits passed from one local computation to the next was determined.

3.3.2 Oracle Separations

The original motivation for the introduction of leaf languages in [BCS92, Ver93] was the wish to have a uniform oracle separation theorem. Usually when relativized complexity classes are separated, this is achieved by constructing a suitable oracle by *diagonalization,* usually a stage construction. Bovet, Crescenzi, Silvestri, and Vereshchagin wanted to identify the common part of all these constructions in a unifying theorem, such that for future separations, one could concentrate more on the combinatorial questions which are often difficult enough. They showed that to separate two classes defined by leaf languages, it is sufficient to establish a certain non-reducibility between the defining languages. Let $A, B \subseteq \{0,1\}^*$. Say that A is polylogarithmic time bit-reducible to B, in symbols: $A \leq_m^{plt} B$, if there are two functions f, g computable in polylogarithmic time such that for all x, $x \in A \iff f(x,0)f(x,1)\cdots f(x,g(x)) \in B$.

Theorem 3.8 [BCS92, Ver93]. *Let* $A, B \subseteq \{0,1\}^*$. *Then* $A \leq_m^{plt} B$ *if and only if for all oracles* Y, *the inclusion* $(A)^{\mathrm{p}}\mathrm{P}^Y \subseteq (B)^{\mathrm{p}}\mathrm{P}^Y$ *holds.*

Observe that $A \leq_m^{plt} B$ is just another formulation for the containment of A in $(B)^{\mathrm{plog}}\mathrm{POLYLOGTIME}$, which in turn is equivalent to the inclusion of the class $(A)^{\mathrm{plog}}\mathrm{POLYLOGTIME}$ in $(B)^{\mathrm{plog}}\mathrm{POLYLOGTIME}$.

Corollary 3.9. *Let* $A, B \subseteq \{0,1\}^*$. *Then we have:*

$$(A)^{\mathrm{plog}}\mathrm{POLYLOGTIME} \subseteq (B)^{\mathrm{plog}}\mathrm{POLYLOGTIME}$$

if and only if for all oracles Y, *the inclusion*

$$(A)^{\mathrm{p}}\mathrm{P}^Y \subseteq (B)^{\mathrm{p}}\mathrm{P}^Y$$

holds.

In [BS97], Theorem 3.8 was strengthened as follows: It was shown that $(A)^{\mathrm{p}}\mathcal{K} \subseteq (B)^{\mathrm{p}}\mathcal{K}$ for all nontrivial classes $\mathcal{K}$ if and only if A is reducible to B by monotone polylogarithmic-time uniform projection reducibility. Refer to their paper for details.

Observe that a polylogarithmic time bit-reduction cannot (simply because of its time bound) read all of its input. This often allows one to prove $A \not\leq_m^{plt} B$ by an adversary arguments. We give a very simple example.

Example 3.10. Let $E = (0+1)^*1(0+1)^*$, $U = 1^*$ as in Sect. 2. Then $(E)^{\mathrm{p}}\mathrm{P} = \mathrm{NP}$ and $(U)^{\mathrm{p}}\mathrm{P} = \mathrm{coNP}$. Suppose $U \leq_m^{plt} E$. The input $x = 1^n$ must be mapped by this reduction to a word with at least one "1". The computation leading to this "1" however cannot read all of x. If we now define x' by complementing in x a bit which is not queried, then again x' will be mapped to a string in E, which is a contradiction. Thus $U \not\leq_m^{plt} E$, and hence there is an oracle separating coNP from NP.

Vereshchagin in [Ver93] used Theorem 3.8 to establish all relativizable inclusions between a number of prominent classes within PSPACE. His list contains besides the classes of the polynomial time hierarchy also UP, FewP, RP, BPP, AM, MA, PP, IP, and others.

A very satisfactory application of Theorem 3.8 was possible in the following special case. Say that $L \subseteq \Sigma^*$ is a *cardinal language*, if membership in L only depends on the frequency with which the elements of Σ appear in words. This means that if $\Sigma = \{a_1, \ldots, a_k\}$ we can associate L with a set $N(L) \subseteq \mathbb{N}^k$, in such a way that $w \in L$ iff there is a $(v_1, \ldots, v_k) \in N(L)$ where a_i occurs in w exactly v_i times $(1 \leq i \leq k)$. ($N(L)$ is the image of L under the Parikh mapping: $N(L) = \Psi_\Sigma(L)$.) Say that L is of bounded significance if there is a number $m \in \mathbb{N}$ such that for all $(v_1, \ldots, v_k)$ we have

$$(v_1, \ldots, v_k) \in N(L) \iff (\min(v_1, m), \ldots, \min(v_k, m)) \in N(L).$$

Using Ramsey theory, Hertrampf in [Her95a] proved the following:

Theorem 3.11 [Her95a]. *There is an algorithm that, given two cardinal languages A, B of bounded significance, decides if $(A)^{\mathrm{p}}\mathrm{P}^Y \subseteq (B)^{\mathrm{p}}\mathrm{P}^Y$ for all oracles Y.*

Pushing his ideas just a bit further, the following was proved: We say that $p\colon \mathbb{N}^k \to \mathbb{N}$ is a positive linear combination of multinomial coefficients if $p(\vec{v}) = \sum_{u \leq z} \alpha_u \binom{v}{u}$ for some $z \in \mathbb{N}^k$, $\alpha_u \in \mathbb{N}$ (for $u \leq z$, the order taken component-wise).

Theorem 3.12 [CHVW97]. *Let A, B be cardinal languages of bounded significance over a k element alphabet. Then $A \leq_m^{plt} B$ if and only if there are functions $p_1, \ldots, p_k\colon \mathbb{N}^k \to \mathbb{N}$ which are positive linear combinations of multinomial coefficients, such that for all $\vec{v} = (v_1, \ldots, v_k)$, $\vec{v} \in N(A)$ if and only if $(p_1(\vec{v}), \ldots, p_k(\vec{v})) \in N(B)$.*

In other words, if such k functions do not exist, then there is an oracle separating $(A)^{\mathrm{p}}\mathrm{P}$ from $(B)^{\mathrm{p}}\mathrm{P}$. Thus we see that the oracle separation criterion Theorem 3.8 leads to a very strong statement in the context of cardinal languages. This result was used in [CHVW97] to establish a complete list of all relativizable inclusions between classes of the boolean hierarchy over NP and other classes defined by cardinal languages of bounded significance.

Valiant's counting class #P is of course strongly related to the notion of cardinal languages. In the case of #P we just deal with the binary alphabet, and we count the number of "1"s in a leaf string. Closure properties of #P, that is operations that don't lead us out of the class, play an important role to establish inclusions between complexity classes; e.g. Toda's result $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$ [Tod91] and Beigel, Reingold, and Spielman's proof that PP is closed under intersection [BRS91] both heavily build on the fact that #P is closed under certain sums, products, and choose operations.

Similar to Theorem 3.12 one can obtain the following:

Theorem 3.13 [HVW95]. *A function $f : \mathbb{N}^k \to \mathbb{N}$ is a relativizable closure property of* #P *(i.e., relative to all oracles, if $h_1, \ldots, h_k \in \#\mathrm{P}$ then also $f(h_1, \ldots, h_k) \in \#\mathrm{P}$), if and only if f is a positive linear combinations of multinomial coefficients.*

3.3.3 Circuit Lower Bounds

Circuit classes as leaf languages have been considered in [CMTV98, Vol96a]. For background on circuit complexity, we refer the reader to [Str94]. It is immediate from Theorem 3.2 that $(\mathrm{NC}^1)^{\mathrm{p}}\mathrm{P} = \mathrm{PSPACE}$. Additionally one can prove e.g. that $(\mathrm{AC}^0)^{\mathrm{p}}\mathrm{P} = \mathrm{PH}$, and that $(\mathrm{TC}^0)^{\mathrm{p}}\mathrm{P}$ is the counting hierarchy CH, defined in [Wag86b, Wag86a] as $\mathrm{PP} \cup \mathrm{PP}^{\mathrm{PP}} \cup \mathrm{PP}^{\mathrm{PP}^{\mathrm{PP}}} \cup \cdots$. Finer results are given in [Vol98].

Building on leaf language characterizations, the circuit class TC^0 (where we require logtime uniformity) was separated from the counting hierarchy in [CMTV98]. This was improved by Allender [All96] to the following separation.

Theorem 3.14. $\mathrm{TC}^0 \neq \mathrm{PP}$.

Proof sketch. We sketch the proof of the weaker result from [CMTV98]. Suppose that $\mathrm{TC}^0 = \mathrm{CH}$. Then we have $\mathrm{TC}^0 = \mathrm{CH} = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{TC}^0) = \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{CH}) \supseteq \mathrm{EXPTIME}$, thus $\mathrm{P} \supseteq \mathrm{EXPTIME}$, which is a contradiction. Allender now observed that this can be extended to show that any language complete for PP under TC^0 reductions cannot be in TC^0. ❏

In the non-uniform case no similar lower bound for TC^0 is known. If we relax the uniformity condition just a little bit, we know that

$$(\text{logspace-uniform } \mathrm{AC}^0)^{\mathrm{p}}\mathrm{P} = \mathrm{PSPACE}$$

(thus also $\mathrm{BLeaf}^{\mathrm{P}}(\text{logspace-uniform } \mathrm{TC}^0) = \mathrm{PSPACE}$). This shows that logtime-uniformity is critical in the above proof.

In Corollary 3.9 it became clear that the oracle separability of two polynomial time classes is equivalent to the absolute separability of two lower classes with the same acceptance paradigm. A similar relation is known between polynomial time and constant depth circuit classes. E.g. building on previous work by Furst, Saxe, and Sipser [FSS84], Yao in his famous paper used a lower bound for the parity function to construct an oracle separating PSPACE from the polynomial hierarchy [Yao85]. This connection has been exploited a number of times since then.

The formal connection between Theorem 3.8 and the Furst, Saxe, Sipser approach to oracle construction has been given in [Vol98]. The main observation that has to be made is that $\leq_m^{plt}$-reductions can be performed by (uniform) qAC^0 circuits. qAC^0 stands for quasipolynomial AC^0 [Bar92], i.e. unbounded fan-in circuits of constant depth and size $2^{\log^{O(1)} n}$. (Similarly we will also use qTC^0 for quasipolynomial size TC^0 circuits, and qNC^1 for quasipolynomial size NC^1 circuits.)

Theorem 3.15. *Let $A, B \subseteq \{0,1\}^*$. Then we have: $A \notin (B)^{\mathrm{plog}}\mathrm{qAC}^0$ if and only if $(A)^{\mathrm{plog}}\mathrm{qAC}^0 \not\subseteq (B)^{\mathrm{plog}}\mathrm{qAC}^0$ if and only if there is an oracle Y such that $(A)^{\mathrm{p}}\mathrm{PH}^Y \not\subseteq (B)^{\mathrm{p}}\mathrm{PH}$.*

This theorem can be used to attack the "nagging question" [For97] how to separate superclasses of P^{PP} from PSPACE. Some special cases are the following.

Corollary 3.16. *$S_5 \notin \mathrm{qTC}^0$ if and only if $\mathrm{qTC}^0 \neq \mathrm{qNC}^1$ if and only if there is an oracle separating the counting hierarchy from* PSPACE*.*

Proof sketch. Under the assumption $S_5 \in \mathrm{qTC}^0$, the following inclusion chain holds relativizably:

$$\mathrm{PSPACE} = \mathrm{BLeaf}^{\mathrm{P}}(S_5) \subseteq \mathrm{BLeaf}^{\mathrm{P}}(\mathrm{qTC}^0) = \mathrm{CH}.$$

This proves the direction from right to left. For the other direction, if relative to all oracles PSPACE $\subseteq$ CH then S_5 polylogarithmic time bit-reduces to qTC^0, but this class is even closed under qAC^0 reductions. ❑

Define $\mathrm{par} =_{\mathrm{def}} \{ w \in \{0,1\}^* \mid \text{the number of "1"s in } w \text{ is odd} \}$, and let maj be as in Sect. 2.

Corollary 3.17. *$S_5 \notin (\mathrm{maj})^{\mathrm{plog}}(\mathrm{par})^{\mathrm{plog}}\mathrm{qAC}^0$ if and only if there is an oracle separating $\mathrm{PP}^{\oplus\mathrm{P}}$ from* PSPACE*.*

Proof sketch. If $\mathrm{PSPACE} \subseteq \mathrm{PP}^{\oplus\mathrm{P}}$ then S_5 polylogarithmic time bit-reduces to a language in the class $(\mathrm{maj})^{\mathrm{plog}}(\mathrm{par})^{\mathrm{plog}}\mathrm{qAC}^0$, and therefore S_5 is even in this class (it is closed under $\leq_m^{plt}$).

On the other hand, if $S_5 \in (\mathrm{maj})^{\mathrm{plog}}(\mathrm{par})^{\mathrm{plog}}\mathrm{qAC}^0$, then $\mathrm{PSPACE} = \mathrm{BLeaf}^{\mathrm{P}}(S_5)$ $\subseteq \mathrm{BLeaf}^{\mathrm{P}}((\mathrm{maj})^{\mathrm{plog}}(\mathrm{par})^{\mathrm{plog}}\mathrm{qAC}^0) = \mathrm{PP}^{\oplus\mathrm{P}^{\mathrm{PH}}} = \mathrm{PP}^{\oplus\mathrm{P}}$. ❑

A refinement of Theorem 3.15 and further investigations along these lines can be found in [Vol98].

3.4 Definability vs. Tree Shapes

Our quantifier from Sect. 2 coincides as we saw in the polynomial time context with leaf languages for balanced computation trees. The unbalanced case has also attracted some attention in the literature. It was observed in [HVW96] that the relativization result from [BCS92, Ver93] *does not hold* in the case of unbalanced trees. Thus, part of the motivation to consider this construct is gone. Nevertheless definability questions are also interesting in this case. The just mentioned observation even makes a systematic comparison of both models a worthwhile study.

3.4.1 Balanced vs. Unbalanced Trees

In [HVW96] the question of definability of the polynomial hierarchy was addressed. As mentioned earlier in Theorem 3.6, the classes of the log-time hierarchy exactly define the classes of the polynomial hierarchy. However, in the case of unbalanced trees, one can somehow use the tree structure to hide an oracle that is able to count paths. More formally,

Theorem 3.18 [HVW96]. $\mathrm{Leaf}^{\mathrm{P}}(\Sigma_k^{\log}) = (\Sigma_k^{\mathrm{p}})^{\mathrm{PP}}$.

3.4.2 The Acceptance Power of Different Tree Shapes

Hertrampf [Her95b] considered besides the above two models also the definition of classes via leaf languages for computation trees which are *full binary trees.* The obtained classes are noted by $\mathrm{FBTLeaf}^{\mathrm{P}}(\cdot)$. Though trivially for every $B \subseteq \{0,1\}^*$ we have $\mathrm{FBTLeaf}^{\mathrm{P}}(B) \subseteq \mathrm{BLeaf}^{\mathrm{P}}(B) \subseteq \mathrm{Leaf}^{\mathrm{P}}(B)$, Hertrampf proved the somewhat counterintuitive result, that the definability power by arbitrary single regular languages does not decrease but possibly *increases* as the tree shapes get more and more regular; that is for every regular language B there is a regular language B' such that $\mathrm{Leaf}^{\mathrm{P}}(B) = \mathrm{Leaf}^{\mathrm{P}}(B') = \mathrm{BLeaf}^{\mathrm{P}}(B')$, and for every regular language B there is a regular language B' such that $\mathrm{BLeaf}^{\mathrm{P}}(B) = \mathrm{BLeaf}^{\mathrm{P}}(B') = \mathrm{FBTLeaf}^{\mathrm{P}}(B')$.

3.4.3 Definability Gaps

In the case of arbitrary tree shapes, Borchert et al. were able to prove the existence of definability gaps. In particular, the following was shown.

Theorem 3.19. *Suppose the polynomial hierarchy does not collapse, and let B be an arbitrary regular language.*

1. *If* $\mathrm{P} \subseteq \mathrm{Leaf}^{\mathrm{P}}(B) \subseteq \mathcal{K}$, *then* $\mathrm{Leaf}^{\mathrm{P}}(B) = \mathrm{P}$ *or* $\mathrm{Leaf}^{\mathrm{P}}(B) = \mathcal{K}$, *where* $\mathcal{K}$ *is one of the classes* NP, coNP, *or* $\mathrm{Mod}_p\mathrm{P}$ *(for some prime number p)* [Bor94].
2. *If* $\mathrm{NP} \subseteq \mathrm{Leaf}^{\mathrm{P}}(B) \subseteq \mathrm{coUS}$, *then* $\mathrm{Leaf}^{\mathrm{P}}(B) = \mathrm{NP}$ *or* $\mathrm{Leaf}^{\mathrm{P}}(B) = \mathrm{coUS}$ [BKS96] *(analogously for* coNP *and* US*).*

We come back to questions of this kind in Sect. 6.

4 Other Resource Bounds

4.1 Circuit Classes

Corollary 3.7 easily yields the following:

Corollary 4.1. $(S_5)^{\mathrm{p}}\mathrm{AC}^0 = \mathrm{PSPACE}$.

This coincidence between $(\cdot)^{\mathrm{p}}\mathrm{P}$ and $(\cdot)^{\mathrm{p}}\mathrm{AC}^0$ holds under more general circumstances. Let $\mathcal{N}$ denote the set of all languages $L \subseteq \Sigma^*$ that contain a neutral letter e, i.e. for all $u, v \in \Sigma^*$, we have $uv \in L \iff uev \in L$.

Theorem 4.2. *If $B \in \mathcal{N}$ then $(B)^{\mathrm{p}}\mathrm{P} = (B)^{\mathrm{p}}\mathrm{AC}^0$.*

Proof sketch. Correctness of computation paths of nondeterministic Turing machines can be checked in AC^0 as already pointed out in the proof of Corollary 3.7. The required AC^0 computation in input (x, y) now checks that its second input argument is a correct path of the corresponding machine on input x; if so it outputs 1 iff this path is accepting and 0 otherwise. If y does not encode a correct path then the neutral letter is output. ❑

A careful inspection of the just given proof reveals that the result not only holds for language $B \in \mathcal{N}$, $B \subseteq \{0,1\}^*$, but also for languages B that are obtained from some $B' \in \mathcal{N}$, $B \subseteq \Sigma^*$ (possibly $|\Sigma| > 2$) by block encoding. The same generalization holds for all results that we state below for "$B \in \mathcal{N}$" (i.e., Theorem 4.8 and all results in Sect. 5).

In the context of NC^1 and subclasses, some interesting results can be obtained for classes of the form $(\cdot)^{\log}\mathrm{AC}^0$.

First, Barrington's theorem [Bar89] yields:

Theorem 4.3. $(S_5)^{\log}\mathrm{AC}^0 = \mathrm{NC}^1$.

Theorem 4.4.

1. *$(B)^{\log}\mathrm{AC}^0 = \mathrm{NC}^1$ for every regular language B whose syntactic monoid is non-solvable.*
2. *$(\mathrm{SOLVABLE})^{\log}\mathrm{AC}^0 = \mathrm{ACC}^0$.*

Generally the class $(B)^{\log}\mathrm{AC}^0$ roughly corresponds to AC^0 circuits with a B gate on top, e.g. $(\mathrm{maj})^{\log}\mathrm{AC}^0$ is the class of all languages accepted by perceptrons.

AC^0 circuits with arbitrary B gates are examined in [BIS90, BI94] (see also Sect. 6).

4.2 Logspace and Logtime Leaf Languages

In the same spirit as above for nondeterministic *polynomial time* machines, Jenner, McKenzie, and Thérien examined in [JMT94] leaf languages for nondeterministic *logarithmic time* and *logarithmic space* machines.

First turning to the logspace case, we observe that the trivial way to formulate $\mathrm{Leaf}^{\mathrm{L}}(B)$, the class defined by logspace machines with leaf language B, as a class $(\cdot)^{\mathrm{p}}\mathrm{L}$ does not work (L denotes the class of logspace decidable sets). This is because (for $B \in \mathcal{N}$) already $(B)^{\mathrm{p}}\mathrm{P} = (B)^{\mathrm{p}}\mathrm{AC}^0$ (see Sect. 4.1), and therefore also $(B)^{\mathrm{p}}\mathrm{P} = (B)^{\mathrm{p}}\mathrm{L}$.

However, if we turn to logarithmic space-bounded one-way protocol machines or 2-1-machines [Lan86], we can come up with a connection. A *2-1-Turing machine* is a Turing machine with two input tapes: first a (regular) input tape that can be read as often as necessary, and second, an additional (protocol) tape that can be read only once (from left to right). Define 2-1-L to be the class of all two argument languages L that can be computed by logspace-bounded 2-1-TMs such that in the initial configuration, the first argument of the input is on the regular input tape, and the second argument is on the one-way input tape. Then the following can be shown using ideas from [Lan86]:

Theorem 4.5. *Let* $B \subseteq \{0,1\}^*$. *Then* $(B)^{\mathrm{p}}$*2-1*-L $= \mathrm{Leaf}^{\mathrm{L}}(B)$.

Jenner, McKenzie, and Thérien showed that in a lot of cases, the balanced and unbalanced model coincide for logarithmic space machines, and moreover it sometimes coincides with the polynomial time case, e.g. Theorem 3.6 above also holds with leaf languages for logspace machines. Interesting to mention is that in the logarithmic space model, regular leaf languages define the class P, while NC^1 defines the class PSPACE.

In the logarithmic time case, coincidence with the logarithmic time reducibility closure could be shown for all well-behaved leaf languages. Formulated in terms of our quantifier, some of their results read as follows:

Theorem 4.6 [JMT94]. *1.* $(\mathrm{REG})^{\log}\mathrm{DLOGTIME} = \mathrm{NC}^1$.

2. $(\mathrm{CFL})^{\log}\mathrm{DLOGTIME} = \mathrm{LOGCFL}$.

3. $(\mathrm{CSL})^{\log}\mathrm{DLOGTIME} = \mathrm{PSPACE}$.

Theorem 4.7 [JMT94]. *1.* $(B)^{\log}\mathrm{DLOGTIME} = \mathrm{NC}^1$ *for any regular language B whose syntactic monoid is non-solvable.*

2. $(\mathrm{SOLVABLE})^{\log}\mathrm{DLOGTIME} = \mathrm{ACC}^0$.

3. $(\mathrm{APERIODIC})^{\log}\mathrm{DLOGTIME} = \mathrm{AC}^0$.

4.3 Other models

4.3.1 Type 2 Operators

Operators ranging not over words but over oracles, so called *type 2 operators*, have been examined in [BW96, BVW96, VW97] and elsewhere. Most of the considered classes coincide with classes of the form $(B)^{\mathcal{F}}\mathcal{K}$ where $\mathcal{K} = \mathrm{coNP}$ or $\mathcal{K} = \mathrm{PSPACE}$ and $\mathcal{F}$ is the class of all exponential time computable functions (let us write $(B)^{\exp}\mathcal{K}$ as a shorthand for this choice of $\mathcal{F}$). A word of care about the computational model however is in order now. We say that a language L belongs to the class $(B)^{\exp}\mathrm{coNP}$ if there is a function f computable in exponential time, and a set A such that $x \in L \iff \chi_{A_x}[0 \dots f(x)] \in B$, where A is accepted by some co-nondeterministic Turing machine M that on input $\langle x, y\rangle$ runs in time polynomial in the length of x. The length of y is possibly exponential in the length of x; thus to enable M to access all positions of y within its time bound we supply M with a regular input tape on which x is found, and a second input tape for y, which is accessed by an index tape . This special input tape is similar to an oracle tape, and therefore quantifiers over strings on this tape translate to quantifiers over oracles. (In the case of $(B)^{\exp}\mathrm{PSPACE}$ we require our machines to use space no more than polynomial in the length of their regular input x.)

Theorem 4.8. *Let $B \in \mathcal{N}$. Then we have:*

$$(B)^{\exp}\mathrm{EXPTIME} = (B)^{\exp}\mathrm{PSPACE} = (B)^{\exp}\mathrm{coNP}.$$

Proof sketch. If we look at the proof of Theorem 4.2 we see that to check correct computation paths we actually don't need the full power of AC^0. $\Pi_1^{\log}$ is sufficient, but we have to modify the computation model slightly as follows: The log-time machine has a regular input tape (which is accessed as usual by using an index tape) and a second input tape on which the path to be checked is given (again access is by an index tape). We thus get:

$$(B)^{\mathrm{p}}\mathrm{P} = (B)^{\mathrm{p}}\mathrm{AC}^0 = (B)^{\mathrm{p}}\Pi_1^{\log}.$$

Using standard translation arguments we now get the claim of the theorem by lifting up this equation one exponential level. ❑

4.3.2 NC^1 Leaf Languages

In [CMTV98] leaf languages for nondeterministic finite automata were considered. The original input is however first given into a uniform projection, and the result of this projection is then fed into the NFA. Barrington's Theorem 4.3 implies that with regular leaf languages we thus get exactly the class NC^1. Some other characterizations were given in [CMTV98], and the model was also used to examine counting classes within NC^1.

4.3.3 Function Classes

In [KSV97] the definability of function classes has been examined. An oracle separation criterion generalizing Theorem 3.8 was given and applied successfully in some open cases.

5 Leaf Languages vs. Lindström Quantifiers

Lindström quantifiers [Lin66] are a well established generalized quantifier notion in finite model theory. The reader probably has noticed some resemblance of our definition in Sect. 2 with that of Lindström quantifiers. It will be our aim in the upcoming sections to make this precise.

As we will see there is a strong connection between leaf languages for polynomial time machines and *second-order* Lindström quantifiers. Since this notion might not be so well-known, we give—after very briefly recalling some terminology from finite model theory—a precise definition in Sect. 5.1.

In later subsections we will have the need to talk about *the second-order version of a given first-order Lindström quantifier.* We chose to make this precise by talking about the semantics of quantifiers given by *languages* instead of the usual way of defining semantics by classes of structures. In the next subsection, we will define how a language B gives rise to a first-order quantifier Q_B^0 and a second order quantifier Q_B^1.

5.1 Second-Order Lindström Quantifiers

A signature is a finite sequence $\tau = \langle R_1, \dots, R_k, c_1, \dots, c_\ell \rangle$ of relation symbols and constant symbols. A finite structure of signature τ is a tuple $\mathcal{A} = (A, R_1^{\mathcal{A}}, \dots, R_k^{\mathcal{A}}, c_1^{\mathcal{A}}, \dots, c_\ell^{\mathcal{A}})$ consisting of a finite set A (the universe of $\mathcal{A}$) and interpretations of the symbols in τ by relations over $\mathcal{A}$ (of appropriate arity) and elements of $\mathcal{A}$. Struct(τ) is the set of all *finite ordered structures* over τ. The *characteristic string* χ_R of a relation $R \in \{0, \dots, n-1\}^a$ is the string $\chi_R =_{\text{def}} b_1 \cdots b_{n^a}$ where $b_i = 1$ iff the i-th vector in $\{0, \dots, n-1\}^a$ (in the order $(0, \dots, 0, 0) < (0, \dots, 0, 1) < (n-1, \dots, n-1, n-1)$) is in R. For $1 \le i \le n^a$, let $\chi_R[i]$ denote the i-th bit in χ_R.

If $\mathcal{L}$ is a logic (as e.g. FO or SO) and $\mathcal{K}$ is a complexity class, then we say that $\mathcal{L}$ *captures* $\mathcal{K}$ if every property over (standard encodings of) structures decidable within $\mathcal{K}$ is expressible by $\mathcal{L}$ sentences, and on the other hand for every fixed $\mathcal{L}$ sentence ϕ, determining whether $\mathcal{A} \models \phi$ can be done in $\mathcal{K}$. As an abbreviation we will most of the time simply write $\mathcal{K} = \mathcal{L}$.

A first-order formula ϕ with k free variables defines for every structure $\mathcal{A}$ the relation $\phi^{\mathcal{A}} =_{\text{def}} \left\{ \vec{a} \in A^k \;\middle|\; \mathcal{A} \models \phi(\vec{a}) \right\}$, see [EF95].

Every class of structures $K \subseteq \text{Struct}(\sigma)$ over a signature $\sigma = \langle P_1, \dots, \mathrm{P}_s \rangle$ defines the first-order Lindström quantifier Q_K as follows: Let $\phi_1, \dots, \phi_s$ be first-order formulae over signature τ such that for $1 \le i \le s$ the number of free variables in ϕ_i is equal to the arity of P_i. Then

$$Q_K \vec{x}_1, \ldots, \vec{x}_s \left[\phi_1(\vec{x}_1), \ldots, \phi_s(\vec{x}_s)\right]$$

is a Q_KFO formula. If $\mathcal{A} \in \mathrm{Struct}(\tau)$, then

$$\mathcal{A} \models Q_K \vec{x}_1, \ldots, \vec{x}_s \left[\phi_1(\vec{x}_1), \ldots, \phi_s(\vec{x}_s)\right]$$

iff $(A, \phi_1^{\mathcal{A}}, \ldots, \phi_s^{\mathcal{A}}) \in K$.

The just given definition is the original definition given by Lindström [Lin66], which the reader will also find in textbooks, see e.g. [Ebb85, EF95]. For our examinations, the following equivalent formulation will be useful (observe that this only makes sense for ordered structures):

Given a first-order formula ϕ with k free variables and a corresponding finite ordered structure $\mathcal{A}$, this defines the binary string $\chi_{\phi^{\mathcal{A}}}$ of length n^k ($n = |A|$). Now given a sequence $\phi_1, \ldots, \phi_s$ of formulae with k free variables each and a structure $\mathcal{A}$, we similarly get the tuple $(\chi_{\phi_1^{\mathcal{A}}}, \ldots, \chi_{\phi_s^{\mathcal{A}}})$, where $|\chi_{\phi_1^{\mathcal{A}}}| = \cdots = |\chi_{\phi_s^{\mathcal{A}}}| = n^k$. Certainly, there is a one-one correspondence between such tuples and strings of length n^k over a larger alphabet (in our case with 2^s elements) as follows. Let A_s be such an alphabet. Fix an arbitrary enumeration of A_s, i.e. $A_s = \{a_0, a_1, \ldots, a_{2^s-1}\}$. Then $(\chi_{\phi_1^{\mathcal{A}}}, \ldots, \chi_{\phi_s^{\mathcal{A}}})$ corresponds to the string $b_1 b_2 \cdots b_{n^k}$, where for $1 \leq i \leq n^k$, $b_i \in A_s$, $b_i = a_k$ for that k whose length s binary representation (possibly with leading zeroes) is given by $\chi_{\phi_1^{\mathcal{A}}}[i] \cdots \chi_{\phi_s^{\mathcal{A}}}[i]$. In symbols: $w_s(\chi_{\phi_1^{\mathcal{A}}}, \ldots, \chi_{\phi_s^{\mathcal{A}}}) = b_1 b_2 \cdots b_{n^k}$.

This leads us to the following definition: A sequence $[\phi_1, \ldots, \phi_s]$ is in *first-order word normal form,* iff the ϕ_i have the same number k of free variables. Let Γ be an alphabet such that $|\Gamma| \geq 2^s$, and let $B \subseteq \Gamma^*$. Then $\mathcal{A} \models Q_B \vec{x} \left[\phi_1(\vec{x}), \ldots, \phi_s(\vec{x})\right]$ iff $w_s(\chi_{\phi_1^{\mathcal{A}}}, \ldots, \chi_{\phi_s^{\mathcal{A}}}) \in B$.

It can be shown [BV98, Bur96] that every Lindström quantifier Q_K can without loss of generality be assumed to be of the form Q_B as just defined. This is the case since for every sequence $[\phi_1, \ldots, \phi_s]$ of first-order formulae we find an equivalent sequence in word normal form such that the corresponding formulae with Lindström quantifier express the same property.

Second-order Lindström quantifiers are defined as follows [BV98, Bur96]: Given a formula ϕ with free second-order variables $P_1, \ldots, P_m$ and a structure $\mathcal{A}$, define $\phi^{2^{\mathcal{A}}} =_{\mathrm{def}} \left\{ (R_1^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}}) \;\middle|\; \mathcal{A} \models \phi(R_1^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}}) \right\}$, and let $\chi_{\phi^{2^{\mathcal{A}}}}$ be the corresponding characteristic string, the order of vectors of relations being the natural one induced by the underlying order of the universe. If the arities of $P_1, \ldots, P_m$ are $r_1, \ldots, r_m$, resp., then the length of $\chi_{\phi^{2^{\mathcal{A}}}}$ is $2^{n^{r_1} + \cdots + n^{r_m}}$

Let $\sigma = \langle \sigma_1, \ldots, \sigma_s \rangle$ be a signature, where $\sigma_i = \langle P_{i,1}, \ldots, P_{i,m_i} \rangle$ for $1 \leq i \leq s$. Thus σ is a signature consisting of a sequence of s signatures with only predicate symbols each. Let $\ell_{i,j}$ be the arity of $P_{i,j}$. A *second-order structure* of signature σ is a tuple $\mathcal{A} = (A, \mathcal{R}_1, \ldots, \mathcal{R}_s)$, where for every $1 \leq i \leq s$, $\mathcal{R}_i \subseteq \left\{ (R_{i,1}, \ldots, R_{i,m_i}) \;\middle|\; R_{i,j} \subseteq A^{\ell_{i,j}} \right\}$. Given now a signature τ and second-order formulae $\phi_1(\vec{X}_1), \ldots, \phi_s(\vec{X}_s)$ over τ where for every $1 \leq i \leq s$ the number and arity of free predicates in ϕ_i corresponds to σ_i. Let $\mathcal{K}$ be a class of second-order structures over σ. Then $Q_{\mathcal{K}} \vec{X}_1, \ldots, \vec{X}_s \left[\phi_1(\vec{X}_1), \ldots, \phi_s(\vec{X}_s)\right]$ is a $Q_{\mathcal{K}}$SO formula. If $\mathcal{A} \in \mathrm{Struct}(\tau)$, then $\mathcal{A} \models Q_{\mathcal{K}} \vec{X}_1, \ldots, \vec{X}_s \left[\phi_1(\vec{X}_1), \ldots, \phi_s(\vec{X}_s)\right]$ iff $(A, \phi_1^{2^{\mathcal{A}}}, \ldots, \phi_s^{2^{\mathcal{A}}}) \in \mathcal{K}$.

Again, we want to talk about second-order Lindström quantifiers defined by languages. Thus we define analogously to the above: A sequence $\left[\phi_1(\vec{X}_1),\dots,\phi_s(\vec{X}_s)\right]$ of second-order formulae is in *second-order word normal form*, if the $\phi_1,\dots,\phi_s$ have the same predicate symbols, i.e. in the above terminology $\sigma_1 = \cdots = \sigma_s = \langle P_1,\dots,P_m\rangle$. Let for $1 \le i \le m$ the arity of P_i be r_i. Observe that in this case, $|\chi_{\phi_1^{2^{\mathcal{A}}}}| = \cdots = |\chi_{\phi_s^{2^{\mathcal{A}}}}| = 2^{n^{r_1}+\cdots+n^{r_m}}$ (for $n = |A|$), thus $(\chi_{\phi_1^{2^{\mathcal{A}}}},\dots,\chi_{\phi_s^{2^{\mathcal{A}}}})$ corresponds to a word of the same length over an alphabet of cardinality 2^s. Given now a language $B \subseteq \Gamma^*$ with $|\Gamma| \ge 2^s$, the second-order Lindström quantifier given by B is defined by $\mathcal{A} \models Q_B^1\vec{X}\left[\phi_1(\vec{X}),\dots,\phi_s(\vec{X})\right]$ iff $w_s(\chi_{\phi_1^{2^{\mathcal{A}}}},\dots,\chi_{\phi_s^{2^{\mathcal{A}}}}) \in B$.

Again it was shown in [BV98, Bur96] that for every second-order Lindström quantifier $Q_{\mathcal{K}}$ there is an equivalent Q_B^1.

When talking about the first-order Lindström quantifier given by B, we sometimes explicitly write Q_B^0 instead of Q_B. In addition to the above logics Q_B^0FO and Q_B^1SO where we allow Lindström quantifiers followed by an arbitrary first-order (second-order, resp.) formula, we also need Q_B^1FO (where we have a second-order Lindström quantifier followed by a formula with no other second-order quantifiers), and FO(Q_B^0) and SO(Q_B^1) (where we have first-order (second-order, resp.) formulae with arbitrary nesting of universal, existential, and Lindström quantifiers). For a class of languages $\mathcal{C}$ we use the notation $Q_{\mathcal{C}}$ with the obvious meaning, e.g. FO$(Q_{\mathcal{C}}^0)$ denotes all first-order sentences with arbitrary quantifiers Q_B^0 for $B \in \mathcal{C}$.

5.2 A Logical Characterization of the Leaf Concept

The main technical connection between polynomial time leaf languages and second-order Lindström quantifiers is given in the following theorem:

Theorem 5.1. *Let M be a polynomial time nondeterministic machine whose computation tree is always a full binary tree, and let $B \subseteq \{0,1\}^*$. Then there is a Σ_1^1 formula ϕ such that*

$$\mathrm{Leaf}^M(B) = Q_B^1\vec{X}\left[\phi(\vec{X})\right].$$

Proof sketch. We use a modification of Fagin's proof [Fag74]. The Q_B^1 quantifier will bind the nondeterministic guesses of the machine. The second-order quantifiers in ϕ will bind variables Y that encode computation paths of M. The formula $\phi(X)$ says "there is a Y encoding a correct computation path of M corresponding to nondeterministic guesses X, which is accepting." ❑

If we deal with $B \subseteq \Gamma^*$ not necessarily over the binary alphabet, then instead of ϕ above, we get formulae $\phi_s,\dots,\phi_s$ such that

$$\mathrm{Leaf}^M(B) = Q_B^1\vec{X}\left[\phi_1(\vec{X}),\dots,\phi_s(\vec{X})\right].$$

$\phi_i(X)$ says "there is a Y encoding a correct computation path of M corresponding to nondeterministic guesses X, and the leaf symbol produced on this path has a 1 in bit position i (in binary). Thus what we have here is some block-encoding of Γ in binary strings of length s.

The just given theorem shows that $\mathrm{FBTLeaf}^{\mathrm{P}}(B) \subseteq Q_B^1\Sigma_1^1$. The question now of course is if there is a logic capturing $\mathrm{FBTLeaf}^{\mathrm{P}}(B)$. For the special case $B \in \mathcal{N}$, the answer is yes.

Theorem 5.2 [BV98]. *Let $B \in \mathcal{N}$. Then $Q_B^1\mathrm{FO} = \mathrm{BLeaf}^{\mathrm{P}}(B)$.*

Proof sketch. This time Q_B^1 binds the nondeterministic guesses X as well as the encoding Y of a possible computation path. The first order formulae "output" the neutral letter, if Y does not encode a correct path. This proves the direction from right to left. For the other inclusion, we observe that we can design a Turing machine which branches on all possible assignments for the relational variables and then simply evaluates the first-order part. ❑

In the preceding theorem $\mathrm{BLeaf}^{\mathrm{P}}(B)$ is captured by the logic $Q_B^1\mathrm{FO}$ *uniformly* in the sense of [MP93, MP94]; this means that the particular formula describing the Turing machine is independent of the leaf language.

Let us next address the question if the quantifier in the preceding theorem is genuinely second-order. First, we have to give some definitions. A succinct representation [Wag86a, BLT92, Vei96] of a binary word x is a boolean circuit giving on input i the ith bit of x. The succinct version sA of a language A is the following: Given a boolean circuit describing a word x, is $x = x_1 0 x_2 0 \cdots x_{n-1} 0 x_n 1 w$ for arbitrary $w \in \{0,1\}^*$, such that $x_1 x_2 \cdots x_n \in A$? The boolean circuits we allow are standard unbounded fan-in circuits over AND, OR, NOT. The encoding consists of a sequence of tuples (g, t, h), where g and h are gates, t is the type of g, and h is an input gate to g (if g is not already an input variable).

Now we see that there is an equivalent first-order logic for $Q_B^1\mathrm{FO}$.

Theorem 5.3. *Let $B \in \mathcal{N}$. Then $\mathrm{BLeaf}^{\mathrm{P}}(B) = Q_B^1\mathrm{FO} = Q_{sB}^0\mathrm{FO}$.*

Proof sketch. Veith [Vei96] showed that sB is complete for $\mathrm{BLeaf}^{\mathrm{P}}(B)$ under projection reductions. (A somewhat weaker result appeared in [BL96]). This together with Theorem 5.2 implies the theorem. ❑

5.3 Applications

Burtschick and Vollmer in [BV98] also examined logically defined leaf languages. It turned out that if the leaf language is given by a first-order formula, then the obtained complexity class is captured by the corresponding second-order logic. More specifically, they proved for instance:

Theorem 5.4 [BV98]. *Let $B \in \mathcal{N}$. Then $(Q_B^0\Sigma_k^0)^{\mathrm{P}}\mathrm{P} = Q_B^1\Sigma_k^1$.*

As a special case of Theorem 5.4 we get a characterization of the classes of the polynomial hierarchy which is tighter than the one in Theorem 3.6.

Corollary 5.5. $(\Sigma_k^0)^{\mathrm{p}}\mathrm{P} = \Sigma_k^{\mathrm{p}}$.

From the PSPACE characterization Theorem 3.2 and the above results, we get the following model-theoretic characterization of PSPACE:

Corollary 5.6. $Q^1_{S_5}\mathrm{FO} = Q^0_{sS_5}\mathrm{FO} = \mathrm{PSPACE}$.

5.4 First-order quantifiers

It is known from the work of Immerman et al. [Imm89, BIS90] that (uniform) AC^0 is captured by FO. However, for this result, we have to include the bit predicate in our logic. We make this assumption throughout this subsection (all the previously given results are valid without the bit predicate).

Theorem 5.7. *Let* $B \subseteq \{0,1\}^*$. *Then* $(B)^{\log}\mathrm{AC}^0 = Q^0_B\mathrm{FO}$.

Theorem 5.7, together with results from Sect. 4.2 on logtime leaf languages, gives some more model-theoretic characterizations.

Corollary 5.8. *1.* $\mathrm{PSPACE} = Q^0_{\mathrm{CSL}}\mathrm{FO} = \mathrm{FO}(Q_{\mathrm{CSL}})$.

2. $\mathrm{LOGCFL} = Q^0_{\mathrm{CFL}}\mathrm{FO} = \mathrm{FO}(Q_{\mathrm{CFL}})$.

Proof sketch. One can show that generally $\mathrm{Leaf}^{\mathrm{LT}}(B) \subseteq Q^0_B\mathrm{FO}$. The corollary then follows from Theorem 4.6. ❑

6 Conclusion

We examined a generalized quantifier notion in computational complexity. We proved that not only all quantifiers examined so far (whether in the logarithmic, polynomial, or exponential time context) can be seen as special cases of this quantifier, but also circuits with generalized gates and Turing machines with leaf language acceptance.

Most of the emerging complexity classes can be characterized by means from finite model theory. We gave a precise connection to finite model theory by showing how complexity classes defined by the generalized quantifier relate to classes of finite models defined by logics enhanced with Lindström quantifiers.

A number of questions remain open. The results we gave in Sect. 5 related complexity classes to logics of the form "Lindström quantifier followed by a usual first- or second-order formula." It is not clear if logics defined by arbitrary nesting of Lindström quantifiers have a nice equivalent in terms of the generalized complexity theoretic quantifier. Barrington, Immerman, and Straubing proved:

Theorem 6.1 [BIS90]. *Let* $B \in \mathcal{N}$. *Then* $\mathrm{FO}(Q^0_B) = \mathrm{AC}^0[B]$ *(*AC^0 *circuits with* B *gates).*

Moreover one can show:

Theorem 6.2 [Vol96b]. *Let $B \in \mathcal{N}$. Then* $\mathrm{FO}(Q_B^1)$ *is the oracle hierarchy given by* $(B)^{\mathrm{p}}\mathrm{AC}^0$ *as building block.*

But the general relationship remains unclear. The work of Makowsky and Pnueli (see [MP93, MP94]), Stewart (see e.g. [Ste91, Ste92]), and Gottlob (see[Got95]) shows that there is a strong relation between Lindström logics and relativized computation. The just mentioned results also hint in that direction. Gottlob [Got95] related the expressive power of logics of the form "Lindström quantifier Q followed by first-order formula" to the expressive power of $\mathrm{FO}(Q)$. However his results only apply for superclasses of L (logarithmic space). Interesting cases within NC^1 remain open. Generally the connection between prenex Lindström logics vs. logics allowing arbitrary quantifier nestings on the model theoretic side, and leaf languages vs. oracle computations on the complexity theoretic side should be made clearer.

It is open for which of the results in Sect. 5.4 the bit predicate is really needed. One can show that without bit, $Q_{\mathrm{CFL}}\mathrm{FO} = \mathrm{CFL}$ contrasting the corresponding result with bit given in Corollary 5.8. The power of the bit predicate in this context deserves further attention.

From a complexity theoretic point of view, we think the main open question is the following. A lot of classes defined by leaf languages have been identified. However, most of the results are not about singular leaf languages but about classes of leaf languages. For example (see Theorem 3.3), if we take an arbitrary aperiodic leaf languages, then the complexity class we obtain is included in PH, and conversely we get all of PH when we allow aperiodic leaf languages: $\mathrm{BLeaf}^{\mathrm{P}}(\mathrm{APERIODIC}) = \mathrm{PH}$. The question now is the following: What exactly are the classes of the form $\mathrm{BLeaf}^{\mathrm{P}}(B)$ for aperiodic B? Is it possible to come up with a complete list of classes that can be defined in this way? Some of the results in Sect. 3.4.3 point in this direction. For example we know that there is no class between P and NP that can be defined by a regular leaf language (unfortunately the result given in Sect. 3.4.3 holds only for the unbalanced case). Can we come up with similar result for the balanced case? Generally, very little is known about the power of *single* leaf languages as opposed to classes of leaf languages.

Acknowledgment. For helpful discussions I am grateful to J. Makowsky (Haifa) and H. Schmitz (Würzburg).

References

[All96] E. Allender. A note on uniform circuit lower bounds for the counting hierarchy. In *Proceedings 2nd Computing and Combinatorics Conference*, volume 1090 of *Lecture Notes in Computer Science*, pages 127–135. Springer Verlag, 1996.

[Bar89] D. A. Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[Bar92] D. A. Mix Barrington. Quasipolynomial size circuit classes. In *Proceedings 7th Structure in Complexity Theory*, pages 86–93. IEEE Computer Society Press, 1992.

[BC94] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* International Series in Computer Science. Prentice Hall, London, 1994.

[BCS92] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[BDG95] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 2nd edition, 1995.

[BI94] D. A. Mix Barrington and N. Immerman. Time, hardware, and uniformity. In *Proceedings 9th Structure in Complexity Theory*, pages 176–185. IEEE Computer Society Press, 1994.

[BIS90] D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[BKS96] B. Borchert, D. Kuske, and F. Stephan. On existentially first-order definable languages and their relation to NP. Technical Report MATH-AL-11-1996, Institut für Algebra, Technische Universität Dresden, 1996.

[BL96] B. Borchert and A. Lozano. Succinct circuit representations and leaf language classes are basically the same concept. *Information Processing Letters*, 58:211–215, 1996.

[BLT92] J. L. Balcázar, A. Lozano, and J. Torán. The complexity of algorithmic problems on succinct instances. In R. Baeza-Yates and U. Manber, editors, *Computer Science.* Plenum Press, New York, 1992.

[Bor94] B. Borchert. On the acceptance power of regular languages. In *Proceedings 11th Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 449–460. Springer Verlag, 1994.

[BRS91] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. In *Proceedings 23rd Symposium on Theory of Computing*, pages 1–9. ACM Press, 1991.

[BS97] B. Borchert and R. Silvestri. The general notion of a dot operator. In *Proceedings 12th Conference on Computational Complexity*, pages 26–44. IEEE Computer Society Press, 1997.

[BT88] D. A. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the Association of Computing Machinery*, 35:941–952, 1988.

[Bur96] H. J. Burtschick. *Berechnungs- und Beschreibungskomplexität von Zählfunktionen und Lindströmquantoren.* PhD thesis, Fachbereich Informatik, TU-Berlin, 1996.

[BV98] H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.

[BVW96] R. V. Book, H. Vollmer, and K. W. Wagner. On type-2 probabilistic quantifiers. In *Proceedings 23rd International Colloquium on Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 369–380. Springer Verlag, 1996.

[BW96] H. Baier and K. W. Wagner. The analytic polynomial-time hierarchy. Technical Report 148, Institut für Informatik, Universität Würzburg, 1996.

[CC95] L. Cai and J. Chen. On input read-modes of alternating turing machines. *Theoretical Computer Science*, 148:33–55, 1995.

[CF91] J.-Y. Cai and M. Furst. PSPACE survives constand-width bottlenecks. *International Journal of Foundations of Computer Science*, 2:67–76, 1991.

[CHVW97] K. Cronauer, U. Hertrampf, H. Vollmer, and K. W. Wagner. The chain method to separate counting classes. *Theory of Computing Systems*, 1997. To appear.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 1998. To appear. Preliminary version in *Proceedings 11th Computational Complexity*, pages 12–21, IEEE Computer Society Press, 1996.

[Ebb85] H.-D. Ebbinghaus. Extended logics: The general framework. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter II, pages 25–76. Springer Verlag, New York, 1985.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer Verlag, Berlin Heidelberg, 1995.

[Eil76] S. Eilenberg. *Automata, Languages, and Machines*, volume B. Academic Press, New York, 1976.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computations*, pages 43–73, Providence, RI, 1974. American Mathematical Society.

[For97] L. Fortnow. Counting complexity. In A. Selmand and L. A. Hemaspaandra, editors, *Complexity Theory Retrospective II*, pages 81–107. Springer Verlag, New York, 1997.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[Got95] G. Gottlob. Relativized logspace and generalized quantifiers over finite structures. Technical Report CD-TR-95/76, Institut for Information Systems, Vienna University of Technology, 1995. An extended abstract appeared in *Proceedings 10th Symposium on Logic in Computer Science*, pages 65–78, IEEE Computer Society Press, 1995.

[Her95a] U. Hertrampf. Classes of bounded counting type and their inclusion relations. In *Proceedings 12th Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 60–70. Springer Verlag, 1995.

[Her95b] U. Hertrampf. Regular leaf-languages and (non-) regular tree shapes. Technical Report A-95-21, Institut für Mathematik und Informatik, Medizinische Universität zu Lübeck, 1995.

[Her97] U. Hertrampf. Acceptance by transformation monoids (with an application to local self reductions). In *Proceedings 12th Conference on Computational Complexity*, pages 213–224. IEEE Computer Society Press, 1997.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[HVW95] U. Hertrampf, H. Vollmer, and K. W. Wagner. On the power of number-theoretic operations with respect to counting. In *Proceedings 10th Structure in Complexity Theory*, pages 299–314, 1995.

[HVW96] U. Hertrampf, H. Vollmer, and K. W. Wagner. On balanced vs. unbalanced computation trees. *Mathematical Systems Theory*, 29:411–421, 1996.

[Imm89] N. Immerman. Expressibility and parallel complexity. *SIAM Journal on Computing*, 18:625–638, 1989.

[JMT94] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. In *Proceedings 9th Structure in Complexity Theory*, pages 242–254, 1994.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 67–161. Elsevier, 1990.

[KSV97] S. Kosub, H. Schmitz, and H. Vollmer. Uniformly defining complexity classes of functions. Technical Report 183, Institut für Informatik, Universität Würzburg, 1997.

[Lan86] K. J. Lange. Two characterizations of the logarithmic alternation hierarchy. In *Proceedings 12th Symposium on Mathematical Foundations of Computer Science*, volume 233 of *Lecture Notes in Computer Science*, pages 518–526. Springer Verlag, 1986.

[Lin66] P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[MP93] J. A. Makowsky and Y. B. Pnueli. Oracles and quantifiers. In *Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 189–222. Springer Verlag, 1993.

[MP94] J. A. Makowsky and Y. B. Pnueli. Logics capturing relativized complexity classes uniform. In D. Leivant, editor, *Logic and Computational Complexity*, volume 1995 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[RS97] R. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume I. Springer Verlag, 1997.

[RV97] K. Regan and H. Vollmer. Gap-languages and log-time complexity classes. *Theoretical Computer Science*, 188:101–116, 1997.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings 5th ACM Symposium on the Theory of Computing*, pages 1–9, 1973.

[Ste91] I. A. Stewart. Comparing the expressibility of languages formed using NP-complete operators. *Journal of Logic and Computation*, 1:305–330, 1991.

[Ste92] I. A. Stewart. Using the Hamilton path operator to capture NP. *Journal of Computer and System Sciences*, 45:127–151, 1992.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[Thé81] D. Thérien. Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14:195–208, 1981.

[Tod91] S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.

[Tor91] J. Toran. Complexity classes defined by counting quantifiers. *Journal of the Association for Computing Machinery*, 38:753–774, 1991.

[Vää94] J. Väänänen. A short course on finite model theory. Lecture Notes, 1994.

[Vei96] H. Veith. Succinct representation, leaf languages, and projection reductions. In *Proceedings 10th Structure in Complexity Theory*, pages 118–126. IEEE Computer Society Press, 1996.

[Ver93] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[Vol96a] H. Vollmer. Relations among parallel and sequential computation models. In *Proceedings 2nd Asian Computing Science Conference*, volume 1179 of *Lecture Notes in Computer Science*, pages 23–32. Springer Verlag, 1996.

[Vol96b] H. Vollmer. Succinct inputs, Lindström quantifiers, and a general complexity theoretic operator concept. Technical Report 158, Institut für Informatik, Universität Würzburg, 1996.

[Vol98] H. Vollmer. Relating polynomial time to constant depth. *Theoretical Computer Science*, 207:159–170, 1998.

[VW97] H. Vollmer and K. W. Wagner. On operators of higher types. In *Proceedings 12th Conference on Computational Complexity*, pages 174–184. IEEE Computer Society Press, 1997.

[Wag86a] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[Wag86b] K. W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, 1986.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

[Yao85] A. C. C. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings 26th Foundations of Computer Science*, pages 1–10. IEEE Computer Society Press, 1985.